10장. 추상클래스와 인터페이스

1. 추상 클래스

10.1.1. 추상 클래스 선언

추상함수

- 추상함수(Abstract method) 는 미완성함수 혹은 실행영역을 가지지 않는 함수
- 추상클래스(Abstract Class)는 추상 함수를 가지는 클래스

```
abstract class AbstractTest1 {
    fun myFun1() {
        //.....
    }
    abstract fun myFun2()
}
```

- 추상함수는 클래스 내부에 선언된 함수에서만 선언할수 있으며 Top-Level에 선언된 함수에는 abstract 예약어를 추가할수 없다.

추상 프로퍼티

```
abstract class AbstractTest2 {
    val data1: String = "kkang"
    abstract val data2: String
}
```

10.1. 추상 클래스

10.1.2. 추상 클래스 이용

- 추상 클래스는 객체 생성이 불가
- 추상 클래스를 상속받아 하위 클래스를 작성하고 하위 클래스를 객체 생성해서 이용

```
abstract class Super {
    val data1: Int = 10
    abstract val data2: Int

    fun myFun1() {

    }
    abstract fun myFun2()
}

class Sub: Super() {
    override val data2: Int = 10
    override fun myFun2() {

    }
}

fun main(args: Array<String>) {
    val obj1=Super()//error
    val obj2=Sub()
}
```

10.2. 인터페이스

10.2.1. 인터페이스 선언 및 구현

- 인터페이스는 추상 함수를 선언함을 주 목적으로 사용
- 인터페이스는 객체생성이 불가
- 클래스에서 인터페이스를 구현해 이용

```
interface MyInterface {
    var data1: String
    fun myFun1() {
        //.............
    }
    fun myFun2()
}

class MyClass: MyInterface {
    override var data1: String = "hello"
    override fun myFun2() {
        //.............
    }
}

fun main(args: Array<String>) {
    val obj=MyInterface()//error
    val obj1=MyClass()
    obj1.myFun1()
    obj1.myFun2()
}
```

10.2. 인터페이스

다른 인터페이스를 상속받는 인터페이스

```
interface MyInterface1 {
    fun myFun1()
}
interface MyInterface2 {
    fun myFun2()
}

interface MyInterface3: MyInterface1, MyInterface2 {
    fun myFun3()
}

class MyClass1: MyInterface3 {
    override fun myFun1() { }
    override fun myFun2() { }
    override fun myFun3() { }
}
```

10.2. 인터페이스

클래스에서 여러 인터페이스 구현

```
interface MyInterface4 {
    fun myFun4()
}
interface MyInterface5 {
    fun myFun5()
}

class MyClass4: MyInterface4, MyInterface5 {
    override fun myFun4() { }
    override fun myFun5() { }
}
```

클래스에서 상속과 인터페이스 혼용

```
interface MyInterface6 {
    fun myFun6()
}
interface MyInterface7 {
    fun myFun7()
}
open class Super {

}
class Sub: Super(), MyInterface6, MyInterface7 {
    override fun myFun6() { }
    override fun myFun7() { }
}
```

10.2. 인터페이스

객체 타입으로서의 인터페이스

```
interface MyInterface10 {
    fun myInterfaceFun()
}
open class Super1 {
    fun mySuperFun(){
        println("mySuperFun()....")
    }
}
class Sub1: Super1(), MyInterface10 {
    override fun myInterfaceFun() {
        println("myInterfaceFun cal....")
    }
}
fun main(args: Array<String>) {
    val obj1: Sub1 = Sub1()
    val obj2: Super1 = Sub1()
    val obj3: MyInterface10 = Sub1()
    //Sub1 타입 객체 이용.....
   obj1.mySuperFun()
    obj1.myInterfaceFun()
    //Super1 타입 객체 이용.....
   obj2.mySuperFun()
    obj2.myInterfaceFun()//error
    //"MyInterface10 타입 객체 이용....."
   obj3.mySuperFun()//error
    obj3.myInterfaceFun()
}
```

10.2. 인터페이스

10.2.2. 인터페이와 프로퍼티

- 인터페이스내에 함수 이외에 프로퍼티도 추가 가능

- 추상형으로 선언되어 있거나 get(), set() 를 정의해 주어야 한다.
- 추상 프로퍼티가 아니라면 val 의 경우는 get()이 꼭 선언되어 있어야 한다.
- 추상 프로퍼티가 아니라면 var 의 경우는 get(), set() 이 꼭 선언되어 있어야 한다.
- 인터페이스의 프로퍼티를 위한 get(), set() 에서는 field를 사용할수 없다.

```
interface MyInterface8 {
    var prop1: Int // abstract
    val prop2: String = "kkang"//error

    val prop2: String//error
        get() = field

    var prop3: String//error
        get() = "kkang“

    val prop4: String
        get() = "kkang"

    var prop5: String
        get() = "kkang"
        set(value) {

        }
}
```

10.2. 인터페이스

- get(), set() 내부에서 field 사용이 불가능 하지만 일정정도의 필요 로직을 추가 가능

```
interface MyInterface9 {

    var data1: Int

    var data2: Int
        get() = 0
        set(value){
            if(value > 0)
                calData(value)
        }

    val data3: Boolean
        get(){
            if(data1 > 0) return true
            else return false
        }


    private fun calData(arg: Int) {
        data1 =  arg * arg
    }

}
```

10.2. 인터페이스

10.2.3. 오버라이드 함수 식별

- 상속받은 클래스에 정의된 함수명과 인터페이스에 정의된 함수명이 중복되는 경우

동일 이름의 추상함수가 여러 개인 경우

```
interface Interface1 {
    fun funA()
}

interface Interface2 {
    fun funA()
}

open abstract class Super2 {
    abstract fun funA()
}

class Sub2: Super2(), Interface1, Interface2 {
    override fun funA() {
        println("Sub2 funA...")
    }
}

fun main(args: Array<String>) {
    val obj1=Sub2()
    obj1.funA()
}
```

10.2. 인터페이스

동일 이름의 추상함수와 구현 함수가 여러 개인 경우

```
interface Interface3 {
    fun funA(){
        println("Interface3 funA....")
    }
}

open abstract class Super3 {
    abstract fun funA()
}

class Sub3: Super3(), Interface3 {
    override fun funA() {
        super.funA()
        println("Sub3 funA....")
    }
    fun some(){
        super.funA()
    }
}

fun main(args: Array<String>) {
    val obj1=Sub3()
    obj1.funA()
    obj1.some()
}
```

10.2. 인터페이스

동일 이름의 구현 함수가 여러 개인 경우

```
interface Interface4 {
    fun funA() {
        println("Interface4 funA...")
    }
}
interface Interface5 {
    fun funA() {
        println("Interface5 funA...")
    }
}

class Sub4: Interface4, Interface5 {//error

}
```

```
class Sub4: Interface4, Interface5 {
    override fun funA() {
        super<Interface4>.funA()
    }
}

fun main(args: Array<String>) {
    val obj=Sub4()
    obj.funA()
}
```

10.2. 인터페이스

```
class Sub4: Interface4, Interface5 {
    override fun funA() {
        super<Interface4>.funA()
        super<Interface5>.funA()
    }
    fun some(){
        super<Interface5>.funA()
    }
}
fun main(args: Array<String>) {
    val obj=Sub4()
    obj.funA()
    obj.some()
}
```